IT Offshoring: Command and Delivery

Mastering IT offshoring to India with onshore command centres: the thin, senior layer that keeps intent, architecture and acceptance at home while execution scales offshore.

Abstract

Offshoring IT work to India fails far more often than the talent on either side deserves, and it fails for a reason that is organisational, not geographical. Firms offshore the wrong thing: they send out responsibility along with execution, cutting the work loose from the intent, the architecture and the judgement that make it correct, and then measure the result in head-count saved rather than value delivered. The distance and the nine-to-twelve-hour time difference, which could be leverage, become pure latency, and the rework they cause quietly eats the labour-cost arbitrage that was the whole point. We argue for a specific operating model that repairs this: a thin, senior *onshore command centre*, called a *stateside command centre* in American practice, that retains the small set of things that must not be offshored, deciding what to build and why, owning the architecture and the interfaces, converting ambiguous business intent into unambiguous specification, holding acceptance and the relationship with the business, while execution scales in India. We give a simple model of where the line between the two should fall, turning on a task's *context load* and the *specification completeness* the command centre can give it, and we derive an optimal *command ratio* of onshore control to offshore delivery: neither throwing work over the wall ($r \to 0$) nor keeping it all at home ($r \to \infty$) is right, and the cost-minimising ratio is finite, positive and computable. Done this way, distance stops being a tax and the clock becomes a second shift. A constructed case traces the numbers. This is practitioner-facing argument, not a benchmark.

Keywords: IT offshoring; India; global software development; command centre; onshore–offshore; global delivery model; coordination cost; Conway's law; follow-the-sun; specification; outsourcing; distributed teams; captive centre; time-zone arbitrage; rework; software architecture

1. Introduction

For two decades the offer has been the same and the arithmetic has looked irresistible. An engineer in India costs a fraction of one in Western Europe or the United States, the supply of them is large and well-trained, and English is the working language, so a firm that moves its software work there should, on paper, do the same work for much less. A great deal of work has moved on exactly that reasoning, and India has built a world-class IT services industry on it. And yet the results, firm by firm, are strikingly uneven: some organisations get excellent software from India for years, while others get a rising bill, a falling quality of delivery, and a stream of defects that the saved salaries never quite cover, until they declare offshoring a failure and bring the work home, often to conclude that the problem was India. It was not India.

The problem is that firms offshore the wrong thing. The intuition behind offshoring is that software development is *execution*, a quantity of work that can be measured in person-hours and bought wherever person-hours are cheapest, and on that intuition a firm hands over a specification and a head-count and waits for output. But software development is not mainly execution; it is the resolution of a thousand small ambiguities, each of which requires knowing what the business actually meant, how this part is supposed to fit the whole, and what "done" really means here, and that knowledge lives where the business is, not where the keyboards are. When a firm offshores execution it unknowingly offshores those decisions too, to the people furthest from the context needed to make them well, and the result is work that is locally plausible and globally wrong, discovered late, across a twelve-hour gap, and fixed slowly. The labour arbitrage is real; the coordination and rework it triggers, handled badly, are larger.

This paper is about the operating model that fixes the mistake by not making it, and the model has a name we will use throughout: the *onshore command centre*. The idea is to separate, deliberately and structurally, the two things that a naive offshoring conflates. One is execution, the building, which genuinely can and should scale where talent is abundant and affordable, in India. The other is *command*: deciding what to build and in what order, owning the architecture and the interfaces between parts, turning what the business vaguely wants into a specification precise enough to be built far away without a conversation, and judging whether what came back is right. Command is small, senior and expensive, and it stays onshore, next to the business whose intent it exists to carry. The thesis is that the success of an offshoring programme is determined almost entirely by whether this command layer exists and is good, and hardly at all by the offshore team's raw ability, which is usually fine; and that firms fail by under-investing in command precisely because command is the expensive, onshore part that offshoring was supposed to eliminate.

The contribution is an argument and a small model, not a benchmark. Section 2 places the claim against the record of offshoring and global software engineering. Section 3 diagnoses the real failure, offshoring responsibility rather than merely execution. Section 4 describes the command centre and what it owns. Section 5 gives a model of where the line between command and delivery should fall, turning on context load and specification completeness. Section 6

treats the clock, and how follow-the-sun becomes leverage rather than latency. Section 7 derives the optimal command ratio. Section 8 traces a constructed case; Sections 9 to 12 draw the consequences, answer objections, state limits and conclude. The question throughout is: **what must a firm keep at home, and how much of it, so that the work it sends to India comes back right?**

2. Background

2.1. The offshoring record

The outsourcing and offshoring of IT has one of the larger empirical literatures in information systems, and its verdict is not that offshoring does not pay but that its payoff is contingent on how it is managed. Lacity, Khan and Willcocks (2009), reviewing two decades of studies, find that client and supplier capabilities, contract design and relationship management, rather than wage differences alone, explain the spread between successful and failed engagements. Carmel and Tjia (2005) gave the practitioner account of sourcing to a global workforce and catalogued the ways distance erodes the arbitrage; Sahay, Nicholson and Krishna (2003) gave the book-length account of software development across the India boundary, and Oshri, Kotlarsky and Willcocks (2007) traced how a leading Indian vendor organised a global delivery model around dispersed expertise. Rottman and Lacity (2006) distilled proven practices and found that the firms that succeeded invested heavily in the onshore work of specifying, coordinating and governing, the very work this paper calls command. The consistent finding is that the saving is gross, not net, and that what stands between the two is management effort that largely stays at home.

2.2. Why distance is expensive

A second literature, on globally distributed software development, explains the mechanism. Herbsleb and Mockus (2003), in the study most central to this paper, showed empirically that work items requiring people at multiple sites took substantially longer to complete than comparable single-site items, and traced the delay to the cost and latency of the communication that coordination across sites requires. Ramasubbu and Balan (2007), studying projects dispersed between India and the United States, measured the same penalty directly in project performance. Olson and Olson (2000) gave the general result under the title "distance matters": collocated work benefits from rapid, rich, informal communication that distributed work loses, and no tooling fully restores it. Kraut and Streeter (1995) had already shown how much of software development is coordination rather than coding, and Brooks (1975) had long before given the reason adding people to distributed work can slow it: communication paths grow faster than the people do. The lesson is not that distributed development cannot work but that every boundary a piece of work must cross has a price, and the price is paid in time and defects.

2.3. Conway's law and the architecture of the boundary

The deepest idea the paper leans on is Conway's (1968): a system's structure comes to mirror the communication structure of the organisation that builds it. Its corollary for offshoring is exact and under-used. If offshore and onshore are two loosely-communicating groups, then the software will be cleanest when the boundary between them falls along a genuine architectural seam, a module with a defined interface, and will be most troubled where a single tightly-coupled lump of work is split across the gap. Herbsleb and Grinter (1999) confirmed this directly, finding that distributed development succeeds when the architecture is modular enough that each site owns a part and the interfaces between parts are explicit, and struggles when it is not. Parnas (1972) gave the design discipline, information hiding, that makes such modules possible. The practical consequence, developed in Section 4, is that owning the architecture is not one of the command centre's jobs among many; it is the job that makes all the others possible, because it decides where the expensive boundary falls.

3. The real failure: offshoring responsibility

The characteristic offshoring failure is not slow typing; it is work that is confidently finished and quietly wrong. A requirement is handed over that is complete in the sender's head and ambiguous on the page, because the sender shares a context with the business that the receiver does not. The offshore engineer, competent and willing, resolves the ambiguity the only way available from twelve time zones away: by guessing, reasonably, and building. The guess is discovered to be wrong at acceptance, if there is acceptance, or in production, if there is not, and the correction crosses the gap twice, once to explain and once to verify, each crossing a day lost to the clock. Multiply this by every ambiguity in a real system and the arbitrage is gone, not because anyone did anything incompetent, but because the decisions that needed context were made where there was none.

What has happened, in the language of this paper, is that the firm offshored *responsibility* when it meant to offshore only *execution*. Responsibility for a piece of software is the accumulated authority to resolve its ambiguities correctly: to know what the business meant, to decide how the part serves the whole, to say what "correct" is. That authority is inseparable from context, and context is local. Execution, the construction of a thing once its ambiguities are resolved, is not context-bound and travels well. The error is to treat the two as one parcel and ship them together; the repair is to split them, keep responsibility onshore where its context lives, and send offshore only execution, with its ambiguities already resolved into specification. Everything that follows is an elaboration of how to make that split and where.

4. The command centre

The onshore command centre is the organisational home of the responsibility that must not travel. It is deliberately small and deliberately senior, because its leverage comes from the quality of a few decisions rather than the quantity of output, and it sits next to the business, not next to the keyboards. Figure 1 draws the model and Table 3 divides the labour. The command centre owns five things, and the discipline of the model is that it owns all five and tries to own nothing else.

Intent to specification. The command centre's first product is not software but specification: the translation

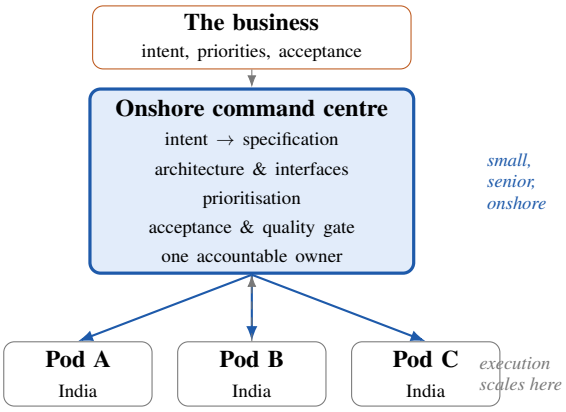

Figure 1. The operating model. A thin, senior onshore command centre sits between the business and the offshore delivery pods. It converts the business's intent into specification, owns the architecture that decides where the onshore–offshore boundary falls, holds prioritisation and acceptance, and carries single-point accountability. Execution scales in the pods; responsibility does not leave the command centre.

of what the business vaguely wants into an account precise enough to be built by someone who cannot lean over and ask. This is the work that resolves ambiguity where the context is, so that none has to be resolved where it is not, and it is the single highest-leverage activity in the whole model, because every ambiguity left in a specification is paid for later at the full cost of a round trip across the gap.

Architecture and interfaces. By Conway's law the software will fracture along the organisation's communication seams, so the command centre owns the architecture precisely in order to choose where those seams fall: it decomposes the system into modules with explicit interfaces so that each offshore pod owns a part it can build with few cross-boundary dependencies. Drawing the boundary well is what converts distance from a tax paid on every interaction into a tax paid only at a few well-defined interfaces.

Prioritisation. What gets built first is a business judgement, not a delivery one, and it stays with the command centre, which holds the single backlog and its order. Dispersing prioritisation to the offshore team, or to several stakeholders at once, reintroduces across the gap exactly the coordination the model exists to contain.

Acceptance and the quality gate. The command centre decides what "done" is and tests the returned work against it, holding a gate the business trusts. This is the counterpart of specification: having defined correctness up front, it verifies it on return, and because it is onshore and close to the business it can judge fitness, not merely function.

One accountable owner. Finally the command centre is where accountability is undivided: a single owner answerable to the business for what is delivered, who cannot point across the gap. Accountability that is split across the boundary is accountability that has gone missing, and restoring it to one onshore point is much of what makes the model governable.

Table 1. Where to draw the line. The command centre's leverage is to move high-context work rightward by raising specification completeness until the residual ambiguity α is small enough to cross the gap.

	Low specification completeness	**High specification completeness**
Low context load	Offshore; light specification suffices	Offshore cleanly; the ideal case
High context load	*The failure zone*: guesswork at distance	Specify first, then offshore with confidence

5. Drawing the line

If command stays onshore and execution goes offshore, everything turns on where the line between them is drawn, task by task. The model that governs the choice has two axes. The first is a task's *context load*: how much local, tacit, business-specific knowledge resolving it correctly requires, high for a change that touches a subtle regulatory rule or a long-standing customer commitment, low for a well-bounded component with a clear contract. The second is the *specification completeness* the command centre can give it: how fully its ambiguities can be resolved in advance, on paper, without a conversation. A task is safe to offshore when the residual ambiguity that reaches the offshore pod, its context load net of the specification completeness applied to it, is small. Writing κ for context load and s for specification completeness, both on a common scale, the residual is

$$\alpha = \max(\kappa - s,\ 0), \tag{1}$$

Residual ambiguity reaching the offshore pod: context load net of the specification applied.

and it is α, not the raw difficulty of the work, that predicts rework. The model has a sharp managerial reading, drawn in Table 1. A low-context task offshores cleanly whatever its technical difficulty. A high-context task is *not* unsuitable for offshoring; it is unsuitable for offshoring *unspecified*, and the command centre's job is to raise s until α is small enough to send. Where context load is genuinely irreducible, where the knowledge cannot be written down at all, the task stays onshore, and recognising that honestly is part of the discipline. The failure mode the model names is the high-context task shipped with a low-completeness specification: maximum residual ambiguity, resolved offshore by guesswork, at maximum distance.

6. The clock

The time difference between a Western business and India, some nine to twelve hours, is the feature practitioners most often treat as a pure liability and the one the command centre can most cleanly convert into an asset. Untended, the gap is latency: a question asked at the end of the offshore day is answered at the end of the onshore day, and a single clarification costs a calendar day whatever its substance, so a chain of clarifications costs a week to resolve an hour of real content. This is the mechanism by which badly-run offshoring feels slow even when everyone works hard: the

critical path runs through the gap, and the gap adds a day per crossing.

Run well, the same gap is a second shift. If the command centre hands off work whose ambiguities are already resolved, the offshore pods execute through the onshore night and return results by the onshore morning, and the day the business sleeps through is a day of progress rather than a day of waiting. The difference between the two outcomes is entirely the residual ambiguity α of what is handed off: a handoff with $\alpha \approx 0$ needs no round trip and the clock is pure gain; a handoff full of unresolved questions turns the clock against you, because each question is a full day. A crude way to see it is that the effective speed-up from the second shift is

$$S \approx 1 + \eta\,(1-\alpha), \tag{2}$$

The second shift pays off in proportion to how unambiguous the handoff is; ambiguity turns the clock into delay.

where η measures how cleanly work can be handed off and α is the residual ambiguity of equation (1); as $\alpha \to 0$ the clock approaches a genuine extra shift, and as α rises the gain collapses and then reverses into the day-per-question tax. The command centre earns the clock by owning the specification and the definition of done that make α small, which is why follow-the-sun is a consequence of good command, not a scheduling trick that can be bought separately.

7. The command ratio

How much command is enough? The question has a real answer, and it is neither "as little as possible", the instinct that produces throw-it-over-the-wall offshoring, nor "as much as possible", which rebuilds onshore the cost that offshoring was meant to shed. Let r be the *command ratio*, the capacity of the onshore command centre as a fraction of the offshore delivery capacity it governs. Command costs the high onshore rate w_c per unit; delivery costs the low offshore rate w_o per unit. The value of command is that it lowers the residual ambiguity reaching the pods and so the rework they must do: more command means more work specified, reviewed and gated before and after it crosses the gap, and so a lower rework multiplier on the delivered work. A serviceable form for that multiplier is

$$m(r) = 1 + a\,e^{-br}, \tag{3}$$

The rework multiplier on offshore delivery falls as command rises: more specification and acceptance, less guesswork.

where a scales the rework that no command at all would incur, set by the intrinsic ambiguity of the work, and b measures how effectively command converts into resolved ambiguity. The total cost of delivering one unit of finished work, command plus reworked delivery, is then

$$C(r) = w_c\,r \;+\; w_o\left(1 + a\,e^{-br}\right), \tag{4}$$

Cost per unit of finished work: onshore command rises linearly in r; offshore rework falls as command resolves ambiguity.

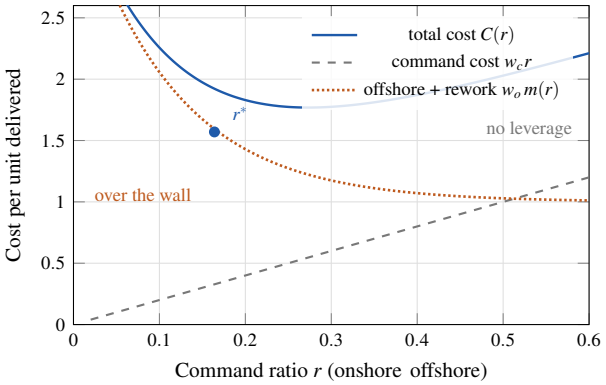

Figure 2. The command ratio has an interior optimum. Too little command (left) leaves offshore delivery to resolve ambiguity by guesswork, and rework dominates; too much (right) rebuilds the onshore cost that offshoring was meant to shed. The cost-minimising ratio r^* of equation (5) balances the two. Values illustrative.

a sum of a rising term, the cost of command, and a falling one, the rework that command prevents, so it has a single interior minimum. Setting $C'(r) = 0$ gives the cost-minimising command ratio

$$r^* = \frac{1}{b}\ln\!\left(\frac{w_o\,a\,b}{w_c}\right), \tag{5}$$

The optimal command ratio: it rises with offshore ambiguity and the leverage of command, and falls as command grows expensive relative to delivery.

which is the paper's central quantitative result. It says three things a manager can act on. The optimal amount of command is *positive*: $r^* > 0$ whenever $w_o a b > w_c$, that is, whenever command is at all effective and ambiguity at all real, so throwing work over the wall ($r = 0$) is never optimal. It is *finite*: command past r^* costs more than the rework it saves, so keeping everything onshore is not optimal either. And it *moves the right way*: r^* rises with the ambiguity of the work (a) and with how well command resolves it (b), and falls as onshore command grows expensive relative to offshore delivery (w_c/w_o), so ambiguous, high-context portfolios warrant a thicker command layer and simple, well-bounded ones a thinner one. Figure 2 draws $C(r)$ and the two failure regimes on either side of the optimum.

8. A constructed case

The case is constructed, not observed: its figures are illustrative and internally consistent, and it is meant to show the model's quantities taking plausible values rather than to report a result. A Western group runs application development for its back-office systems and, three years ago, offshored the bulk of it to a vendor in India on a staff-augmentation basis: it kept a handful of onshore project managers, wrote requirements in the form it always had, and measured success by the head-count moved and the day-rate saved. On paper it had halved the cost of a developer. In practice the straight-through delivery of a feature, from request to accepted production change, had lengthened, the defect-escape rate into production had risen, and a large and growing share

Table 2. Application delivery, before and after a command centre. The offshore team and its rates are unchanged; what changes is the residual ambiguity reaching it and the rework that follows. Illustrative figures for the constructed group.

	Over the wall	**Command centre**
Onshore share of head-count	4%	12%
Feature lead time (request to prod)	rising	cut by half
Defect escape into production	high	low, gated
Offshore hours lost to rework	∼35%	∼10%
Residual ambiguity α	high	low
Command ratio r	≈ 0.04	≈ 0.16
Net saving vs onshore	thin, falling	large, stable

Table 3. The division of labour. Responsibility, which is bound to local context, stays onshore; execution, which is not, scales offshore. The model fails when items migrate across this line.

Stays onshore (command)	**Scales offshore (delivery)**
Deciding what to build and why	Building to a clear specification
Architecture and interfaces	Implementation within a module
Intent → specification	Component and unit testing
Prioritisation; the single backlog	Working the backlog in order
Acceptance; the quality gate	Fixing against acceptance feedback
Accountability to the business	Accountability for delivery to spec

of the offshore team's hours went into reworking features that had been built to a reasonable but wrong reading of an ambiguous requirement. The saving was gross; net, it was thin and falling. Table 2 gives the before column.

The group then rebuilt the engagement around a command centre. It did not change vendor or move work home; it added a thin, senior onshore layer, a few per cent of the total head-count, that took ownership of the five things of Section 4: it rewrote requirements as specifications with explicit acceptance criteria, re-cut the architecture so that each offshore pod owned a module behind a defined interface, held a single ordered backlog, and ran an acceptance gate before anything reached production. The offshore team and its rates were unchanged; what changed was the residual ambiguity α reaching it. Rework fell as specifications resolved the guesswork that had driven it, the defect-escape rate dropped as the gate caught what the gap used to hide, and the clock turned from latency into a second shift as handoffs stopped needing round trips. The cost of the added command was a fraction of the rework it removed, which is equation (5) in practice: the engagement had been running far to the left of r^*, and moving toward it lowered total cost even as it raised onshore cost. Table 2 traces the change.

9. What changes for the operating model

Adopting the command centre changes what a firm must be good at, and the change is uncomfortable because it runs against the instinct that motivated offshoring. The saving is still real, but it is captured only by spending, onshore and at senior rates, on the command layer that the naive business case treated as waste to be eliminated. Table 3 sets out the division of labour the model requires. The scarce resource becomes the handful of senior people who can hold intent, architecture and acceptance well, and a firm's offshoring capacity is bounded by how many such people it has, not by how many offshore engineers it can hire. The metric changes too: head-count moved and day-rate saved, the numbers that make throw-it-over-the-wall offshoring look good, are replaced by feature lead time, defect escape and the share of offshore hours spent on rework, the numbers that reveal whether command is doing its job.

There is a second shift, in where seniority sits. Throw-it-over-the-wall offshoring tends to send the interesting, senior work offshore along with the rest and keep a junior coordination layer at home; the command-centre model inverts that, keeping the seniority onshore where the context is and letting the offshore team be excellent at execution without being asked to supply context it cannot have. This is often misread as distrust of the offshore team. It is the opposite: it is the arrangement that lets a capable offshore team do its best work, by not asking it to resolve ambiguities that no one at its distance could resolve well, and by giving it specifications clear enough that its competence is spent on building rather than on guessing.

10. Objections and replies

The command centre just adds onshore cost and erodes the saving. It adds onshore cost and *raises* the net saving, which is the whole point of equation (5): most failing engagements sit far to the left of r^*, where a little more command removes much more rework than it costs. The command centre is not overhead on the saving; it is the mechanism that converts a gross saving into a net one. A firm genuinely already at r^* should not add command; almost none are.

If you must specify everything so precisely, why not just build it onshore? Because specification and construction have very different costs and very different scarcities. Specification is a small quantity of senior, context-bound work; construction is a large quantity of work that is not context-bound once specified. The model sends the large, cheap-to-locate quantity to where it is affordable and keeps only the small, context-bound quantity at home. Collapsing the two back together forgoes exactly the arbitrage that is real.

Modern tooling and video calls have made distance irrelevant. They have made it smaller, not irrelevant, and the evidence that distance still matters is robust (Olson & Olson, 2000; Herbsleb & Mockus, 2003). Tooling lowers the cost of a crossing; it does not remove the day the clock adds to a round trip, nor restore the informal, ambient communication that collocated teams use to resolve ambiguity without noticing. The command centre is the structural answer that tooling complements but does not replace.

Our offshore partner is very capable and can own the whole thing. Capability is not the issue; context is. A highly capable offshore team still cannot resolve an ambiguity whose resolution requires knowing what the business meant, and asking it to do so sets its competence against a problem

distance makes insoluble. The model keeps responsibility onshore not because the offshore team is weak but because responsibility is bound to a context that is onshore.

11. Limitations

This is a practitioner-facing argument with a deliberately simple model, and its limits should be plain. The model of Sections 5 to 7 is illustrative: the residual-ambiguity form of equation (1), the exponential rework multiplier of equation (3) and the single optimal ratio of equation (5) are simplifications chosen for legibility, and the constants a, b, w_c and w_o are not measured here; a firm can estimate them from its own rework and rate data, and we would expect the qualitative shape, an interior optimum that moves with ambiguity and relative cost, to survive even where the exact form does not. The argument is scoped to work whose ambiguities can, with effort, be resolved into specification; genuinely research-like work, where no one yet knows what correct is, fits the model badly and should mostly stay collocated. The paper draws on India because that is where the depth of supply and the scale of the practice are, but nothing in the model is specific to a country; it applies to any onshore–offshore split across a real distance and clock. And the model assumes a firm able to staff a senior command layer at all; a firm that cannot should be honest that it is not yet able to offshore well, rather than offshoring badly and calling the result a verdict on India.

12. Conclusion

Offshoring IT work to India disappoints firms that offshore the wrong thing: they ship out responsibility along with execution, cutting the work loose from the context and judgement that make it correct, and then watch rework and the clock eat the saving they came for. The repair is not to bring the work home but to split it correctly, keeping onshore the thin, senior command that must stay next to the business, deciding what to build, owning the architecture and the interfaces, turning intent into specification, and holding acceptance, while execution scales in India where talent is abundant and affordable. Where to draw the line is a question the residual ambiguity of a task answers, and the command centre's craft is to raise specification until that ambiguity is small enough to cross the gap; how much command to keep is a question with a real optimum, positive and finite, that rises with the ambiguity of the work and falls as command grows dear. Done this way the distance stops being a tax and the twelve-hour clock becomes a second shift, and the arbitrage that was always real is finally kept. Offshore the building; command it from home. A firm that remembers which of the two is bound to context, and which is not, can master offshoring to India; a firm that forgets will keep concluding, wrongly, that the problem was the other end of the line.

Appendix A. Standing up a command centre

The practices below are what make the split of Section 4 hold in practice. None is novel; the discipline is the value.

Table 4. A command-centre checklist. Each practice keeps responsibility onshore and lowers the residual ambiguity that crosses the gap; the right-hand column is the failure it prevents.

Practice	What it prevents
Staff command with seniority, onshore, near the business	A junior layer guessing at intent
Write specifications with explicit acceptance criteria	Ambiguity resolved offshore by guesswork
Cut the architecture so each pod owns a module behind an interface	Distance taxed on every interaction
Hold one ordered backlog in the command centre	Priorities negotiated across the gap
Run an acceptance gate before production	Wrong work discovered by users
Hand off only low-ambiguity work; keep the irreducible onshore	The clock turning into day-per-question delay
Keep single-point accountability onshore	Accountability lost across the boundary
Measure lead time, defect escape and rework, not headcount saved	Mistaking gross saving for net

References

[1] Brooks, F. P. (1995). *The mythical man-month: Essays on software engineering* (Anniversary ed.). Addison-Wesley. ISBN 978-0-201-83595-3. (Original work published 1975.)

[2] Carmel, E. (1999). *Global software teams: Collaborating across borders and time zones*. Prentice Hall. ISBN 978-0-13-924218-6.

[3] Carmel, E., & Agarwal, R. (2001). Tactical approaches for alleviating distance in global software development. *IEEE Software*, 18(2), 22–29. https://doi.org/10.1109/52.914734

[4] Carmel, E., & Tjia, P. (2005). *Offshoring information technology: Sourcing and outsourcing to a global workforce*. Cambridge University Press. ISBN 978-0-521-84355-4.

[5] Conway, M. E. (1968). How do committees invent? *Datamation*, 14(4), 28–31.

[6] Herbsleb, J. D., & Grinter, R. E. (1999). Splitting the organization and integrating the code: Conway's law revisited. In *Proceedings of the 21st International Conference on Software Engineering (ICSE)* (pp. 85–95). ACM. https://doi.org/10.1145/302405.302455

[7] Herbsleb, J. D., & Mockus, A. (2003). An empirical study of speed and communication in globally distributed software development. *IEEE Transactions on Software Engineering*, 29(6), 481–494. https://doi.org/10.1109/TSE.2003.1205177

[8] Kraut, R. E., & Streeter, L. A. (1995). Coordination in software development. *Communications of the ACM*, 38(3), 69–81. https://doi.org/10.1145/203330.203345

[9] Krishna, S., Sahay, S., & Walsham, G. (2004). Managing cross-cultural issues in global software outsourcing. *Communications of the ACM*, 47(4), 62–66. https://doi.org/10.1145/975817.975818

[10] Lacity, M. C., Khan, S. A., & Willcocks, L. P. (2009). A review of the IT outsourcing literature: Insights for practice. *Journal of Strategic Information Systems*, 18(3), 130–146. https://doi.org/10.1016/j.jsis.2009.06.002

[11] Malone, T. W., & Crowston, K. (1994). The interdisciplinary study of coordination. *ACM Computing Surveys*, 26(1), 87–119. https://doi.org/10.1145/174666.174668

[12] Olson, G. M., & Olson, J. S. (2000). Distance matters. *Human–Computer Interaction*, 15(2–3), 139–178. https://doi.org/10.1207/S15327051HCI1523_4

[13] Oshri, I., Kotlarsky, J., & Willcocks, L. P. (2007). Managing dispersed expertise in IT offshore outsourcing: Lessons from Tata Consultancy Services. *MIS Quarterly Executive*, 6(2), 53–65. https://aisel.aisnet.org/misqe/vol6/iss2/3/

[14] Parnas, D. L. (1972). On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, 15(12), 1053–1058. https://doi.org/10.1145/361598.361623

[15] Ramasubbu, N., & Balan, R. K. (2007). Globally distributed software development project performance: An empirical analysis. In *Proceedings of the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE 2007)* (pp. 125–134). ACM. https://doi.org/10.1145/1287624.1287643

[16] Rottman, J. W., & Lacity, M. C. (2006). Proven practices for effectively offshoring IT work. *MIT Sloan Management Review*, 47(3), 56–63. https://sloanreview.mit.edu/article/proven-practices-for-effectively-offshoring-it-work/

[17] Sahay, S., Nicholson, B., & Krishna, S. (2003). *Global IT outsourcing: Software development across borders*. Cambridge University Press. ISBN 978-0-521-81604-5.